**DOCKET NO. 1999.06.003.WS0**
**Customer No. 23990**

**PATENT**

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

| | | |
|---|---|---|
| In re application of: | : | Steven E. Lovette |
| Serial No. | : | 09/197,993 |
| Filed | : | November 23, 1998 |
| For | : | TECHNIQUE FOR DETECTING CORRUPTION ASSOCIATED WITH A STACK IN A STORAGE DEVICE |
| Group No. | : | 2154 |
| Examiner | : | Dustin Nguyen |

**MAIL STOP APPEAL BRIEF - PATENTS**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

## APPELLANT'S BRIEF ON APPEAL

This Appeal Brief is submitted in triplicate on behalf of Appellant for the above-identified application. The Notice of Appeal was filed in the Patent Office on April 14, 2005. The Appellant respectfully requests a two-month extension of time for filing the Appeal Brief. Please reconsider the application in light of the following arguments, which the Appellant makes in order to more particularly define the issues for appeal.

08/16/2005 MWOLDGE1 00000006 09197993
01 FC:1402 500.00 OP

## REAL PARTY IN INTEREST

The real party in interest for this appeal is the assignee of the application, Samsung Electronics Co., Ltd.

## RELATED APPEALS AND INTERFERENCES

There are no other appeals or interferences related to the present application that are currently pending

## STATUS OF CLAIMS

Claims 1-15 were previously cancelled. Claims 26-49 are pending in the application. Claims 26-49 were rejected under U.S.C. §103(a) as being obvious over U.S. Patent No. 6,006,323 to *Ma et al.* (hereafter, "*Ma*") in view of U.S. Patent No. 5,890,181 to *Selesky et al.* (hereafter, "*Selesky*").

## STATUS OF AMENDMENTS

No amendments were entered subsequent to final rejection.

## SUMMARY OF CLAIMED SUBJECT MATTER

The present invention relates to detection of corruption in a stack caused by an overflow condition and/or an underflow condition. The present invention uses guard frames 44 that are placed in the address locations immediately preceding and immediately following a stack. See Figure 4 and

Specification at page 11, line 14 to page 12, line 12. Each guard frame 44 is placed at an address location that borders the stack, but not in the stack. The guard frame 44 holds a predetermined value. The predetermined value may be a fixed bit pattern, a known address value, or a known instruction value. See Figure 4 and Specification at page 12, lines 1-12. If the guard frame 44 is overwritten during a stack push or a stack pop, the predetermined value will be changed. Thus, whenever a stack push or a stack pop operation occurs, a guard function checks the guard frames 44 to detect corruption in the guard frame. See Figure 4 and Specification at page 13, line 11 to page 14, line 15. If the value in the guard frame 44 does not match the predetermined value, then an overflow condition or an underflow condition has occurred.

## GROUNDS OF REJECTION TO BE REVIEWED UPON APPEAL

Claims 26-49 were rejected under 35 U.S.C. §103(a) as being obvious over the *Ma* and *Selesky* references.

## ARGUMENT

Independent Claims 26 and 38 recite analogous limitations and may be treated as a group. The Appellant directs the Board's attention to Claim 26, which contains the following unique and non-obvious limitations:

> 26. A method for detecting corruption associated with a stack in a storage device, the method comprising the steps of:

storing a first predetermined value in a first address location immediately preceding the starting location of the stack;

detecting the occurrence of a stack operation within the stack; and

comparing the value in the first address location to the first predetermined value to determine if the stack operation corrupted the first predetermined value stored in the first address location. (*emphasis added*)

The Appellant respectfully asserts that the above-emphasized limitations are not disclosed, suggested, or even hinted at in the *Ma* reference or the *Selesky* reference, or in the combination of the *Ma* and *Selesky* references.

The Appellant believes that the Examiner's rejection of Claim 26 may be based in part on a misunderstanding by the Examiner of the difference between: 1) the value of an address, and 2) the value stored in an address location. In Paragraph 3 of the continuation sheet of the Advisory Action, the Examiner stated "Furthermore, Ma discloses comparing the value in the first address location to the first predetermined value [i.e., top of stack address and stack pointer when popped or pushed] [column 1, lines 18-37, and column 2, lines 1-16]." The Appellant respectfully asserts that this statement is factually incorrect, as will be explained below in greater detail.

Regarding the Claim 26 limitation of "[a] method for detecting corruption associated with a stack," the Examiner asserted that the *Ma* reference "discloses a method for detecting corruption [i.e., detect over/underflow]," citing Figure 5, the Abstract, and column 4, lines 10-19. The Appellant respectfully submits that the Examiner has misunderstood the teaching of the *Ma* reference. In fact, the *Ma* reference describes a stack management unit that prevents stack corruption, rather than one that detects corruption associated with a stack, as recited in Claim 26.

The *Ma* reference discloses determining a condition that indicates the potential for corruption of the stack (i.e., the overflow condition) and preventing that corruption by moving adequate data out of a primary stack into a secondary stack to relieve the overflow condition. *See Abstract, last sentence; col. 9, lines 25-36.* If the detection of the overflow condition by the *Ma* system were the detection of corruption, as argued by the Examiner, then the actions of the *Ma* system in response – copying primary stack data into the secondary stack – would result in the copying of corrupted stack data into the secondary stack.

Regarding the Claim 26 limitation of "comparing [a] value in [a] first address location to [a] first predetermined value," the Examiner incorrectly asserted that the *Ma* reference discloses comparing the value in a first address location with a predetermined value, citing column 14, lines 58-62. The Examiner also cited column 13, lines 13-22, of the *Ma* reference as disclosing this limitation. The Appellant respectfully submits that the Examiner has misunderstood the language of Claim 26 and the teachings of the *Ma* reference.

Claim 26 recites a stack in a storage device and a first location, also in the storage device, immediately preceding the start of the stack. Claim 26 further recites examining a value stored in that first location in order to detect corruption associated with the stack. In contrast, the *Ma* reference teaches comparing the locations (i.e., the addresses) of first and last elements stored in a stack for the purpose of sensing an overflow condition and preventing stack corruption. The *Ma* reference clearly describes this functionality at column 8, line 65, through column 9, line 21. "A top of stack (TOS) register 55 . . . indicates the last register written to in the primary stack (e.g., the end

of a data element). A bottom of stack (BOS) register 56 indicates the beginning of the oldest element . . . ." *Ma, col. 8, line 65, through col. 9, line 2.*

The Appellant respectfully asserts that the TOS value is the address of the last used register in the stack, and the BOS is the address of the first used register in the stack. "To calculate the current usage of the primary stack, the TOS value is subtracted from the BOS value by subtractor 57. As shown, the K [least significant bits] of the result are compared to overflow and underflow limitation values . . . ." *Ma, col. 9, lines 16-19.* The Appellant respectfully asserts that neither the TOS value nor the BOS value is a value stored IN an address location. Rather, the TOS and BOS values are pointers that point TO the address location. Thus, the address of the register at the top of the stack is subtracted from the address of the register at the bottom of the stack to determine how many stack registers are currently in use. The difference in address values (not the value stored at a location adjacent the stack, as recited in Claim 26) is then compared to predefined values to sense the overflow condition and thereby prevent stack corruption.

In summary, the *Ma* reference does not teach the method recited in Claim 26. Instead, the *Ma* reference teaches a stack manager for preventing stack corruption by sensing when too little free space remains in the stack and transferring stack data to a secondary stack. The *Ma* reference does not teach comparing the value in a first address location with a predetermined value, as recited in Claim 26. Instead, the *Ma* reference teaches comparing the difference between addresses of the first and last used registers in a stack to predetermined values.

The Appellant submits that the *Selesky* reference does nothing to overcome the shortcomings of the *Ma* reference. The *Selesky* reference was introduced because it purportedly discloses storing a first predetermined value in a first address location immediately preceding the starting location of the stack. The Appellant respectfully asserts that the *Selesky* reference does no such thing. The Examiner relied on items 34a and 36a in Figure 4 of the *Selesky* reference and the text of the *Selesky* reference at column 6, lines 59-65. The Appellant respectfully asserts that Figure 4 of the *Selesky* reference discloses a stack 18 in which actions are stored and further discloses separating these actions within the stack 18 by inserting and removing beginning markers (34a, 36a) and inserting and removing end markers (40a, 42a). However, these markers between actions are used entirely within stack 18. The *Selesky* reference does not disclose that either type of marker is used outside of the stack, either in an address location preceding the start of the stack or in an address location following the end of the stack.

As such, the unique and non-obvious limitations recited in Claim 26 are not disclosed, suggested, or even hinted at in the *Ma* reference or the *Selesky* reference, or in the combination of the *Ma* and *Selesky* references. This being the case, Claim 26 presents patentable subject matter over the *Ma* and *Selesky* references. The Appellant respectfully asserts that independent Claim 38 contains limitations that are analogous to the unique and non-obvious limitations recited in Claim 26.

The Appellant further notes that dependent Claims 32 and 44 also contain limitations that are analogous to the unique and novel limitations recited in Claims 26 and 38, except that Claims 32 and 44 are directed to the detection of stack corruption using values stored in address locations that

immediately follow the bottom of the stack, rather than precede the top of the stack. This being the case, dependent Claims 32 and 44 also present patentable subject matter over the *Ma* and *Selesky* references.

## SUMMARY

For the reasons given above, the Appellant respectfully requests reconsideration and allowance of pending claims and that this Application be passed to issue. If any outstanding issues remain, or if the Examiner has any further suggestions for expediting allowance of this Application, the Appellant respectfully invites the Examiner to contact the undersigned at the telephone number indicated below or at ***jmockler@davismunck.com***.

The Commissioner is hereby authorized to charge any additional fees connected with this communication or credit any overpayment to Deposit Account No. 50-0208.

Respectfully submitted,

DAVIS MUNCK, P.C.

Date: 11 August 2005

John T. Mockler
Registration No. 39,775

P.O. Drawer 800889
Dallas, Texas 75380
Phone: (972) 628-3600
Fax: (972) 628-3616
E-mail: *jmockler@davismunck.com*

## CLAIMS APPPENDIX

The status of the claims is as follows:

1-25. Cancelled.

26. (Previously Presented) A method for detecting corruption associated with a stack in a storage device, the method comprising the steps of:

storing a first predetermined value in a first address location immediately preceding the starting location of the stack;

detecting the occurrence of a stack operation within the stack; and

comparing the value in the first address location to the first predetermined value to determine if the stack operation corrupted the first predetermined value stored in the first address location.

27. (Previously Presented) The method as set forth in Claim 26, wherein the first predetermined value comprises a known bit pattern.

28. (Previously Presented) The method as set forth in Claim 26, wherein the first predetermined value comprises a processor readable address.

29. (Previously Presented) The method as set forth in Claim 26, wherein the first predetermined value comprises a processor readable instruction.

30. (Previously Presented) The method as set forth in Claim 26, wherein the stack operation inserts data in the stack.

31. (Previously Presented) The method as set forth in Claim 26, wherein the stack operation removes data from the stack.

32. (Previously Presented) The method as set forth in Claim 26, further comprising the step of storing a second predetermined value in a second address location immediately following the ending location of the stack.

33. (Previously Presented) The method as set forth in Claim 32, wherein the second predetermined value comprises a known bit pattern.

34. (Previously Presented) The method as set forth in Claim 32, wherein the second predetermined value comprises a processor readable address.

35. (Previously Presented) The method as set forth in Claim 32, wherein the second predetermined value comprises a processor readable instruction.

36. (Previously Presented) The method as set forth in Claim 32, wherein the stack operation inserts data in the stack.

37. (Previously Presented) The method as set forth in Claim 32, wherein the stack operation removes data from the stack.

38. (Previously Presented) A system for detecting corruption associated with a stack, the system comprising:

a processor; and

a storage medium comprising a stack, the stack beginning at a starting location and ending at an ending location, wherein the processor is operable to store a first predetermined value in a first address location immediately preceding the starting location of the stack, detect the occurrence of a stack operation within the stack; and compare the value in the first address location to the first predetermined value to determine if the stack operation corrupted the first predetermined value stored in the first address location.

39. (Previously Presented) The system as set forth in Claim 38, wherein the first predetermined value comprises a known bit pattern.

40. (Previously Presented) The system as set forth in Claim 38, wherein the first predetermined value comprises a processor readable address.

41. (Previously Presented) The system as set forth in Claim 38, wherein the first predetermined value comprises a processor readable instruction.

42. (Previously Presented) The system as set forth in Claim 38, wherein the stack operation inserts data in the stack.

43. (Previously Presented) The system as set forth in Claim 38, wherein the stack operation removes data from the stack.

44. (Previously Presented) The system as set forth in Claim 38, wherein the processor is further operable to store a second predetermined value in a second address location immediately following the ending location of the stack.

45. (Previously Presented) The system as set forth in Claim 44, wherein the second predetermined value comprises a known bit pattern.

46. (Previously Presented) The system as set forth in Claim 44, wherein the second predetermined value comprises a processor readable address.

47. (Previously Presented) The system as set forth in Claim 44, wherein the second predetermined value comprises a processor readable instruction.

48. (Previously Presented) The system as set forth in Claim 44, wherein the stack operation inserts data in the stack.

49. (Previously Presented) The system as set forth in Claim 44, wherein the stack operation removes data from the stack.